

Writing Filters and Tools for L2Global

v0.5

D. Casey
Michigan State University

September 18, 2000

This document outlines the procedure to writing a tool and filter for L2Global. If you have any questions/comments, please contact Dylan Casey (casey@pa.msu.edu).

1 Nomenclature

A routine that evaluates data from the L2 preprocessors and constructs a list of candidate physics objects is called a *tool*. A routine which evaluates lists of candidate objects is called a *filter*. Tools create candidate electrons, photon, taus, muons, etc. Filters ask whether there are candidate objects that satisfy some refined criteria. A specific filter considers only objects of a specific type – electrons, jets, or taus. A generic filter considers any type of object created by a tool – electrons and jets and muons. For instance, the tool EMTool creates a list of electromagnetic objects (EMObj) by matching EM clusters from L2CAL (satisfying some minimum Et value) with preshower clusters from L2CPS and L2FPS and with tracks from L2CTT. The specific filter EMFilter searches the list looking for candidates that satisfy some more stringent (configurable) criteria such as a higher Et cut, isolation cut, emfraction cut, etc. The generic filter MassFilter would consider the generic kinematic properties of the specific objects and cut on the mass value of a combination of them.

2 Creating a Tool

The coding requirements for a tool are the following:

- tools inherit from the template class TFilter<[object type]>
- their only public method is print
- the two required private methods are execute(void) and initialize(void), and a static variable that holds the number of filters created
- all configuration parameters (set by data from the parser) are private

- the constructor must declare the tag for the tool to the parser via
`l2gblworker::TFilter<object type>("[name]",_count)`, e.g.,
`l2gblworker::TFilter<EMObj>("GENEMTOOL",_count)`
- the constructor ought to set default values for all of configuration parameters

The algorithm for a tool is contained in the execute function. Every execute function begins by calling `reset()`, so that the relevant lists are reset to zero. Tools inherit from `TFilter<ObjType>` because they create lists of specific objects – jets, taus, emobjects.

The perl script `makeL2ToolFilterSkeleton` may be run to generate the skeleton code for a L2 tool or filter. The script is contained in the `l2gblworker` package. The script is executed in the following manner:

```
makeL2ToolFilterSkeleton -author "Dylan Casey" -email
"casey@pa.msu.edu" -package "l2gblem" -class "EMTool" -store
"EMObj" -toolname "EMTOOL"
```

Two files are created: `EMTool.hpp` and `EMTool.cpp`. There are notations within the skeleton showing where important pieces of user code must be added. Default values for each of the arguments are used if they are not configured explicitly. See the preamble of the script for the default values of the parameters.

The following is the output for `EMTool.hpp` made by `makeL2ToolFilterSkeleton` when run with the example configuration above:

```
//
// File: EMTool.hpp
// Purpose:
// Created: 1 September 2000 by Dylan Casey
//
// Comments:
//
// Revisions:
//
#ifndef _L2GBLEM_EMTOOL_HPP
#define _L2GBLEM_EMTOOL_HPP
#include "l2base/L2.hpp"
#include "l2gblworker/TFilter.hpp"
#include "l2gblem/EMObj.hpp"

// Start namespace for this package
namespace l2gblem {

//=====
// CLASS : EMTool
//
```

```

/// This is a tool for L2 Global.
/** This is a tool for L2 Global
    @author Dylan Casey (casey@pa.msu.edu)
    @version 0.1 1 September 2000
*/
//=====//
class EMTool : public l2gblworker::TFilter<EMObj> {
public:
    /** Constructor which sets the tool requirements to
        their default values. The default cut values are set
        to allow everything to pass. In this way nothing
        should get thrown away if this tool gets called
        without being initialized due to some bug.
    */
    EMTool(void);
    /** Print out the filter's configuration to the given
        output stream. This function print out the tool's
        configuration to the given output stream.
    */
    void print(std::ostream &ostr);
private:
    /** Executes the filter algorithm. This method is called at
        most once per event and will fill the list with electrons
        from the specified source which satisfy the filter cuts.
    */
    void execute(void);
    /** Initializes the electron filter using the parser data.
        This is the method called by the parser to initialize the
        filter using the configuration data. It will only be
        called when the parser has read a configuration object
        for this electron filter.
        @return true is initialization was successful, false otherwise
    */
    bool initialize(void);
    // Count of the number of filters created so far
    static uint32 _count;
    /***** Put cut parameters here *****/
};

// Constructor inline EMTool::EMTool(void) :
    l2gblworker::TFilter<EMObj>("EMTOOL",_count){
    _count++;
    /*****initialize cut parameters here *****/
}

} // end namespace l2gblm

```

```
#endif // _L2GBLEM_EMTOOL_HPP
```

Here is the corresponding source code:

```
//
// File: EMTool.cpp
// Purpose: Source code for EMTool class
// Created: 1 September 2000 Dylan Casey
//
// Comments:
//
// Revisions:
//
#include "l2gblem/EMTool.hpp"
#include "l2gblem/EMObj.hpp"
#include "l2gblworker/GlobalWorker.hpp"
#include "l2gblworker/GlobalInput.hpp"
#include "l2gblworker/GlobalOutput.hpp"

using namespace l2gblworker;

// Start namespace for this package
namespace l2gblem {

// Declaration of static filtercounter
uint32 EMTool::_count=0;

// Initializes the EMTool class; called by the l2parser
bool EMTool::initialize(void) {

/***** retrieve cut values from L2 parser *****/

return true;
}

// Executes the filter process
void EMTool::execute(void) {

// Reset the filters
reset();

// Include some output that is useful during debugging.
// Right now, the flag can only be toggled by recompiling.
bool debug = false;
if(debug){
    std::cout << "EMTool" << parserID() << " has been called!" << std::endl;
}
```

```

        print(std::cout);

    }

    /***** Put the algorithm here *****/

    // Getting this far means we have a EMObj!
    // Fill in the information and add it to the list of passed objects
    EMObj *storeobj = new EMObj;
    /***** set the values of the EMObj data members here *****/
    addObject(storeobj);
    // adds electron to pass list for the filter

    if(debug) {
        storeobj->print(std::cout);
        std::cout << "Adding EMObj to list for EMTool" << std::endl;
    }
    if(debug) {std::cout << "\nEMTool execute ends here!"<<std::endl;}
}
// Prints out the filter's configuration
void EMTool::print(std::ostream &ostr) {
    ostr << parserID() << " {" << std::endl;
    /**** list the parameter names and their values here ****/
    ostr << " PAR1 = " << _par1 << std::endl;
    ostr << " PAR2 = " << _par2 << std::endl;
    ostr << "}" << std::endl;
}
} // end namespace l2gblworker

```

Finally, here is the code skeleton for the stored object:

```

//
// File: EMObj.hpp
// Purpose:
// Created: 1 September 2000 by Dylan Casey
//
// Comments:
//
// Revisions:
//
#ifndef _L2GBLEM_EMOBJ_HPP
#define _L2GBLEM_EMOBJ_HPP
#include "l2base/L2.hpp"
#include "l2base/io.hpp"
#include "l2utils/Storable.hpp"

```

```

#include DATAHEADER(****L2IOClass****)
// Start namespace for this package
namespace l2gblem {
//=====
// CLASS : EMObj
//
/// Holds the EMObj data and creation utilities
/** The class inherits from the l2io data class
    L2IODataClass to have a place to hold the data that is
    sent to L2 and from the Storable class.
    @author Dylan Casey (casey@pa.msu.edu)
    @version 0.1 1 September 2000
*/
//=====
class EMObj : public ***L2IODataClass***,
              public l2utils::Storable<EMObj> {
public:
/** Constructor
*/
EMObj(void);

protected:

private:

};
// Constructor
inline EMObj::EMObj(void) {

} // end namespace l2gblem
#endif // _L2GBLEM_EMOBJ_HPP

```

3 Creating a Filter

Creating a specific filter is the same as creating a tool. There will be two differences. First, you will hold a pointer to the objects created by the tool that you are filtering. You will initialize the pointer in the initialize script that is called by the parser. The second difference will be in the execute routine. Instead of creating new Storable objects, you will just be adding existing ones to your list of stored objects. Compare the EMTool and EMFilter header files, initialize, and execute routines to see the differences. The files are part of the l2gblem package.

Creating a generic filter requires a modest change to the code created by shell script. You need to inherit from Filter not from TFilter<ObjType>. That way you have access to all the objects created by all the tools so far.

4 Putting Tools and Filters into the Simulation

The tools and filters are part of l2gblworker. To execute them as part of the simulation, you must make several changes to l2gblworker.

1. Add the l2gblworker package to your release area
2. Add an instance of your filter or tool to the list of private data members in GlobalWorker.hpp
3. Add an include reference to the header file for your filter or tool to GlobalWorker.hpp
4. Add tsim_l1l2 and tsim_l2 to your release area. tsim_l2 will not be changed, but it is the package that instantiates GlobalWorker, and since you are modifying the interface to GlobalWorker, you need to recompile tsim_l2.
5. Add the appropriate configuration lines to tsim_l1l2/rcp/l2gblTrigger.conf for the particular trigger(s) and filter(s) you want to run.
6. Rebuild everything in your release area.